OPTIMIZED PROCESSORS AND INSTRUCTION ALIGNMENT

INVENTOR: James Loran Ball
                       542 Blackjack Lane
                       Santa Cruz, CA 95062
                       Citizen of Canada

Assignee: Altera Corporation
                       101 Innovation Drive
                       San Jose, CA 95134

BEYER WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, CA 94704-0778
(510) 843-6200

OPTIMIZED PROCESSORS AND INSTRUCTION ALIGNMENT

James Loran Ball

5 ## BACKGROUND OF THE INVENTION

1. Field of the Invention.

The present invention relates to processors. In one example, the present
10 invention relates to methods and apparatus for optimizing a processor and allowing
flexible instruction alignment.

2. Description of Related Art

15 Conventional processors are configured to support a variety of instructions. In
many instances, processor circuitry is configured to perform particular instructions.
Instructions are often handled sequentially using specially configured subcircuitry. In
one instance, a particular type of instruction, the branch instruction, allows the flow
of execution of a program to change by performing a jump to a non-sequential
20 instruction at a branch target address. Specialized circuitry is often provided to
calculate addresses that are the target of the jump operations for branch instructions.

However, mechanisms for efficiently performing calculations for different
types of instructions are limited. In one example, specialized subcircuitry is
25 configured to specifically calculate multi-byte branch aligned target addresses.
However, the specialized subcircuitry is often not reused for other types of
instructions.

Mechanisms for efficiently optimizing processor cores are limited. It is
30 therefore desirable to provide improved methods and apparatus for optimizing
implementation of processors and processor circuitry. In some instances, alignment
restrictions can be relaxed to allow further processor optimization.

Summary of the Invention

Methods and apparatus are provided for optimizing a processor core. Common processor subcircuitry is used to perform calculations for various types of instructions, including branch and non-branch instructions. Increasing the commonality of calculations across different instruction types allows branch instructions to jump to byte aligned memory address even if supported instructions are multi-byte or word aligned.

In one embodiment, a processor is provided. The processor includes a multiple registers and circuitry. The circuitry is configured to process instructions associated with an instruction set including a branch and non-branch instructions. The instructions each having a multi-byte length. The instructions are accessible at multi-byte aligned addresses. Multi-byte aligned branch instructions are operable to access the instructions at byte aligned addresses.

In another embodiment, a processor is provided. The processor includes multiple registers and circuitry. The circuitry is configured to process branch and non-branch instructions associated with an instruction set. The branch instructions and non-branch instructions include an immediate field. Common subcircuitry is used to process the immediate field associated with one or more branch instructions and one or more non-branch instructions.

In another embodiment, a method for performing an instruction is provided. A branch instruction associated with an address is decoded. The branch instruction has an associated opcode and an immediate value. A branch target address is calculated using the immediate value. The branch target address is determined by using common subcircuitry. The common subcircuitry is operable to calculate a byte-aligned address. The common subcircuitry is also configured to perform nonbranch operations. A jump is performed to the branch target address. The branch target address is multi-byte aligned.

These and other features and advantages of the present invention will be presented in more detail in the following specification of the invention and the accompanying figures, which illustrate by way of example the principles of the invention.

5

Brief Description of the Drawings

The invention may best be understood by reference to the following description taken in conjunction with the accompanying drawings, which illustrate specific embodiments of the present invention.

Figure 1 is a diagrammatic representation showing a processor.

Figure 2 is a diagrammatic representation showing a non-branch instruction.

Figure 3 is a diagrammatic representation showing a branch instruction.

Figure 4 is a diagrammatic representation showing a non-branch instruction performed using common subcircuitry.

Figure 5 is a diagrammatic representation showing a branch instruction performed using common subcircuitry.

Figure 6 is a flow process diagram showing instruction processing.

Figure 7 is a diagrammatic representation showing a programmable chip.

Figure 8 is a diagrammatic representation showing an interconnection fabric.

Figure 9 is a diagrammatic representation showing a technique for implementing the programmable chip.

Figure 10 is a diagrammatic representation depicting a computer system.

Detailed Description of Specific Embodiments

Reference will now be made in detail to some specific embodiments of the invention including the best modes contemplated by the inventors for carrying out the invention. Examples of these specific embodiments are illustrated in the accompanying drawings. While the invention is described in conjunction with these specific embodiments, it will be understood that it is not intended to limit the invention to the described embodiments. On the contrary, it is intended to cover alternatives, modifications, and equivalents as may be included within the spirit and scope of the invention as defined by the appended claims. For example, the techniques of the present invention will be described in the context of particular processors and memory.

However, it should be noted that the techniques of the present invention can be applied to a variety of types of devices. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. The present invention may be practiced without some or all of these specific details. In other instances, well known process operations have not been described in detail in order not to unnecessarily obscure the present invention. Furthermore, techniques and mechanisms of the present invention will sometimes be described in singular form for clarity. However, it should noted that some embodiments can include multiple iterations of a technique or multiple instantiations of a mechanism unless noted otherwise. For example, a processor is used in a variety of contexts. However, it will be appreciated that multiple processors can also be used while remaining within the scope of the present invention.

Conventional processors are configured to support a variety of instructions. Some processors support a general-purpose instruction set such as a reduced instruction set computing (RISC) instruction set, a complex instruction set computing (CISC), or a very long instruction word (VLIW), instruction set. Other more specialized processors such as Digital Signal Processing (DSP) processors or video accelerators may support more specialized instruction sets. Typical processors will perform operations based on the type of instruction and the associated parameters

received. For example, a processor can perform a sequence of add instructions followed by memory access instructions such as load and store.

Instruction sets may include a variety of instructions including arithmetic instructions such as add unsigned or subtract unsigned, logical instructions such logical AND and OR, data transfer instructions such as load word or store word, branch instructions such as branch on equal or branch on not equal, and jump instructions. Some processor architectures allow instructions in an instruction set to vary in length. For example, some instructions may be 16-bits in length instructions while other instructions are 32-bits in length. However, other architectures use instructions that have a fixed length. Using an instruction set including only fixed length instructions greatly simplifies implementation of an instruction set.

In many instances, simplicity leads to efficiency. Because instructions are fixed length, the instructions can be aligned with particular memory addresses. For example, instruction memory is addressed in units of bytes. Any instruction having a start address at any byte address is referred to herein as a byte aligned instruction. Any instruction that has a start address that is an integer multiple of a byte address greater than one is referred to herein as a multiple byte aligned instruction. For example, a specific case of a multiple byte aligned instruction is a word aligned instruction. The word aligned instruction has a start address that is aligned at intervals of four bytes.

Consequently, any word aligned instruction in instruction memory will start at an address of four bytes, eight bytes, 12 bytes, 16 bytes, etc. Because instructions are word aligned, some instructions such as branch instructions are only able to jump to particular word aligned addresses. The branch instructions such as branch on equal or branch on not equal instructions allow a change in the flow of execution of a program by jumping to a nonsequential instruction at a branch target address. In some examples, the branch target address is computed as a function of an offset embedded in the branch instruction and the address of the branch instruction. In other examples, a branch instruction may be given an absolute address. Conventional processors use specialized subcircuitry to calculate a branch target address. Specialized subcircuitry

determines the branch target address using the assumption that the branch target address is multi-byte aligned or word aligned.

The techniques and mechanisms of the present invention recognize that using a specialized subcircuit to determine a multi-byte aligned or a word aligned branch target address is inefficient. The techniques and mechanisms of the present invention minimize hardware costs and maximize performance by increasing the commonality of calculations across different instruction types. For example, subcircuitry used to perform a load or store instruction is also used to perform a branch instruction. By using common subcircuitry, hardware costs are reduced because the same subcircuitry can be reused across different instruction types. No dedicated subcircuit is required for noncommon calculations. By decreasing the amount of circuitry in a processor, increasing processor frequency and performance is typically allowed.

In one example, the techniques of the present invention allow the efficient calculation of a branch target. In typical instances, calculating a branch target involves determining the sign extended value of an immediate field associated with a branch instruction. A processor already has other non-branch instructions that perform calculations using the sign extended value of the same immediate field. However, multi-byte alignment issues have conventionally prevented the use of the same subcircuitry to perform both types of instructions. By choosing the branch offset to be in bytes, the same subcircuitry used to perform the sign extension of the immediate field for non-branch instructions can be reused for branch instructions.

Allowing the reuse of subcircuitry for different instruction types provides other benefits as well. By reusing subcircuitry, support for changes in the instruction set is provided. New instructions can be added and flexibly handled using the techniques and mechanisms of the present invention. For example, when smaller instructions such as 16-bit instructions are introduced into a 32-bit instruction set, the alignment restrictions are typically reduced to match the size of the smallest instruction. Using the techniques and mechanisms of the present invention, the branch instruction will be able to jump to these instructions even if the new instructions are not multi-byte aligned. For example, if instructions in instruction set

were previously word aligned, the new instructions can be byte aligned or half word aligned and the branch instruction will still be able to jump to the appropriate memory location.

5   Without using the techniques of the present invention, existing branch instructions will have to be changed to break compatibility or new branch instructions will have to be added to support the smaller instruction alignment restrictions. By adding a whole set of new branch instructions, additional subcircuitry will have to be introduced, potentially increasing hardware costs. Some conventional processors use

10   multi-byte aligned instructions. Many of the conventional processors are being configured to support smaller instructions. In many cases, new branch instructions are being written simply to handle these particular smaller instructions. The conventional processors are consequently left with different groups of branch instructions. One group of branch instructions handles 16-bit aligned instructions.

15   Another group of similar branch instructions handles 8-bit aligned instructions. These redundant groups of branch instructions in conventional processors are highly inefficient.

           Figure 1 is a diagrammatic representation showing a processor configured to

20   handle an instruction set. A processor 101 includes a register bank with register 111, register 113, and register 115. Processor 101 also includes circuitry 131 for handling various instructions. Circuitry 131 includes subcircuitry 141. Any portion of processor circuitry used to perform instructions in an instruction set is referred to herein as subcircuitry. Processor 101 is connected to memory 151 with memory lines

25   153-161. It should be noted that a processor 101 may also have other configurations and include other components not shown such as and instruction cache, data cache, and were a memory controller. In typical examples, a processor reads instructions sequentially from memory and performs these instructions. The instructions may involve operations on data also loaded from memory and placed in registers for even

30   more efficient access. Performing operations such as arithmetic operations involves using particular hardware within the processor 101. A branch instruction may involve using another piece of hardware within processor 101.

In an overly simplistic example, each instruction is handled by a different subcircuit within the processor 101. For example, a logical AND operation may be performed using subcircuitry 141, register 111, and a particular value provided in an instruction. A parameter used in an instruction to provide information for performing a particular operation is referred to herein as an immediate value. In some instances, immediate values are obtained from memory in a prior instruction.

According to various embodiments of the present invention, branch instructions allow the processing of a sequence of operations to move to a different multi-byte aligned memory location. In one example, instructions are word aligned. The branch instruction allows a branch to a particular word aligned address. However, branches to other addresses are not allowed. In general, it will be appreciated that restricting alignment to multi-byte or word aligned memory locations allows access to a greater range of memory addresses. More specifically, a limited number of bits are available to store the offset associated with a branch. In one simplified example, three bits or eight different values of offset are provided. If no multi-byte alignment is used, the branch target address is restricted to a three bit range of eight bytes in the memory space. However, if word alignment is used, the offset is multiplied by 4. The branch target address range is now expanded to a five bit range or thirty-two bytes in the memory space. In this example, the target address still can only be in 18 memory locations, however the eight specific locations are spread over a range of thirty-two bytes instead of being limited to a range of eight bytes.

Consequently, many processor architectures use multi-byte alignment to expand a branch target address range. Any range of addresses that includes a potential target address for a branch instruction is referred to as a branch target address range. In many processors and ASICs, adding logic and subcircuitry to handle unique instructions is less-expensive than adding logic in a programmable chip to handle unique instructions. Logic in many programmable devices is at a premium and consequently the techniques of the present invention recognize that it is particularly advantageous to reuse subcircuitry for handling different types of instructions even if the branch target address range is decreased.

Figure 2 is a diagrammatic representation showing a conventional instruction. According to various embodiments, the instruction includes an opcode specifying a branch instruction 201, empty fields 203 and 205, and an immediate value 207.

5   According to various embodiments, the opcode for the branch instruction 201 is six bits, empty fields 203 and 205 are each five bits, and the immediate field 207 is 16 bits for a total of 32 bits in the instruction. According to various embodiments, instructions are word aligned. When a processor reaches a particular branch instruction, it determines using the offset where the branch target memory address

10   should be found. In one example, the branch target address is computed by multiplying the branch offset by 4, thereby ensuring word alignment and increasing the branch target address range. The results is then signed extended to the number of bits in the instruction address. The sign extended value is then added to the branch instruction address or the program counter. In one example, the calculation is as

15   follows:

branch_target = program_counter + sign_extend (immediate_field * 4)

The multiplication of the immediate value by four is performed because the

20   units of the branch offset is words and the units of the program counter and branch target are bytes.

Figure 3 is a diagrammatic representation showing a conventional arithmetic instruction. According to various embodiments, an instruction includes an add

25   integer operation code (opcode) 301, the source register 303, the destination register 305, and an immediate field value 307. According to various embodiments, the opcode for the add integer instruction 301 is six bits, register fields 303 and 305 are each five bits, and the immediate field 307 is 16 bits for total of 32 bits in the instruction. In one example, the add integer instruction takes the value of the register

30   identified in the source register field 303, adds it to the sign extended value of the immediate field 307, and writes the result to the register specified by the destination register field 305. In one example, the immediate field is sign extended from 16 bits

to 32 bits, or the length of an instruction. According to various embodiments, the add integer instruction is calculated as follows:

dest_register_value = source_register_value + sign_extend (immediate_field)

Although both the branch instruction and the add integer instruction perform sign extension operations on the immediate field value, the branch instruction multiplies the immediate field value by four before performing the sign extension. The multiplication is performed to allow for word alignment and an expanded target branch address range. However, by requiring a multiplication of four before performing the sign extension, different subcircuitry route is required to perform the operation. In many instances, one subcircuit is specifically configured to sign extend an immediate field and another subcircuit is specifically configured to perform a sign extend of an immediate field multiplied by four. In some examples, multiplication by four is performed by using additional multiplexers and shifters on a base sign extended subcircuit. However, having additional hardware or additional subcircuits is expensive, particularly for programmable chips. Consequently, and the techniques and mechanisms of the present invention allow the reuse of subcircuitry for both types of instructions.

Techniques and mechanisms of the present invention allow the units of branch instructions to be in bytes even though many of the instructions are multiple bytes in size and a required to be aligned on multi-byte addresses.

Figure 4 is a diagrammatic representation showing a branch instruction that allows byte aligned addresses. Fields 401 and 403 remain empty. The operation code for the branch instruction is included in field 407. The branch offset is included in the immediate field 405. However, unlike a conventional branch instruction, the immediate field value is maintained in units of bytes instead of units of words. For example, a conventional instruction would indicate that an offset is thirty-two words in size. The value of 32 would be held in a conventional branch instruction immediate field. Multiplication by four would indicate that the offset is 128 bytes in size. According to various embodiments, 128 is held in the immediate field 405

instead of the value 32. Because the value of 128 is held, no multiplication or other conversion is needed. The target address is calculated by sign extending the offset to the number of bits in the instruction address and adding it to the branch instruction address or program counter. According to various embodiments, the sign extended

5   offset is added to a variation on the program counter value such as the program counter plus four. In one embodiment, the calculation is as follows:

branch_target = program_counter + sign_extend (immediate_field)

10   No multiplication by four is required because the units of the branch are bytes and the units of the program counter value and branch target are bytes.

Figure 5 is a diagrammatic representation showing a typical non-branch instruction. The non-branch instruction includes a source register field 501, the

15   destination register field 503, and immediate value field 505, and an opcode such as an add integer instruction 507. According to various embodiments, the add integer instruction takes the value of the register specified by the source register field 501, adds it to the signed extended value of the immediate field 505, and writes the result to the register specified by the destination register field 503. In one embodiment the

20   calculation is as follows:

dest_register_value = source_register_value + sign_extend (immediate_field)

Because both branch and non-branch operations perform a sign extended

25   operation on the immediate field value, the same subcircuit can be reused for both types of operations. Using the same subcircuitry allows efficient use of hardware resources and reduced processor core sizes.

Figure 6 is a flow process diagram showing a technique for using common

30   subcircuitry for handling both branch and non-branch operations. At 601, a source register value is obtained. At 603, an operation on the source register value and the immediate field value is performed using common subcircuitry. In one example, the common subcircuitry used is circuitry for performing a sign extend operation. At

605, the result is written to the destination register. At 611, the value associated with a program counter is obtained. At 613, and operation is performed on the program counter and the immediate field value using the same subcircuitry noted in 603. At 615, a branch is performed on a byte aligned addresses. It should be noted that although Figure 6 is used to describe one example of implication of a non-branch and a branch instruction, many other examples and configurations are possible. Furthermore, performing instructions may involve a variety of other operations such as loads and stores but are not described here for clarity.

Although the techniques and mechanisms of the present invention or applicable to a variety of different processors including general purpose processors, video accelerators, cryptography accelerators, digital signal processors, microcontrollers, etc., the techniques and mechanisms of the present invention or particularly applicable to soft core processors on programmable chips and programmable chip systems. Logic is often at a premium on programmable chips, so the reduction in core sizes is often particularly beneficial.

Figure 7 is a diagrammatic representation showing one example of a system on a programmable chip that can be used to implement the techniques of the present invention. The system includes a processor core, a hardware accelerator, peripheral devices, and peripheral interfaces. The processor core size can be optimized by reusing subcircuitry to perform both branch and non-branch operations. Peripheral devices and peripheral interfaces are herein referred to as components. The system on a programmable chip includes processor core 701 and an Ethernet component 703 as well as peripheral components UART 711, PIO 713, timer 715, and data memory 717. In some examples, the Ethernet component 103 is instead a Digital Signal Processing (DSP) core, a cryptography accelerator, or a video processor. It should be noted that the system can include both on-chip memory 717 and off-chip memory. In one example, the data memory 717 can support variable latency or fixed latency access. The components are interconnected using an interconnection fabric 707. Any mechanism or logic for connecting components in a system is referred to herein as an interconnection fabric. In one example, the interconnection fabric is a bus. In another example, the interconnection fabric is a secondary side arbitration fabric.

The programmable chip uses different types of resources that can be interchangeably used in different allocations to implement the system on a programmable chip. In one example, the programmable chip uses logic elements for implementing each of the various components on the programmable chip.

Logic elements typically can be implemented using components such as antifuses, static RAM, and EPROMS. Any mechanism on a programmable chip that performs an operation on a given number of input lines to provide one or more outputs based on information programmed is herein referred to as a logic element. Some logic elements are implemented as combinations of look up tables and switches for performing Boolean operations on input lines. In one example, a logic element includes a 16-bit SRAM lookup table (LUT) that can implement an arbitrary 4-input logic function, circuitry that forms a fast carry chain and a fast cascade chain, a register and preset/reset logic for the register.

The techniques and mechanisms of the present invention allow the implementation of a system on a programmable chip from a high-level language program. In one example, variable latency and fixed latency can be supported on a system using a conventional bus architecture.

A system bus typically has a set width (e.g. 64 bits, 128 bits) and allows only one primary component to actively use the bus at any one time. In conventional systems, only one primary component can access any one of the given secondary components at any given time. Multiple primary components accessing secondary components in a manner that would cause data bit collisions if performed on the same data lines is referred to herein as accessing secondary components at the same time.

In one example, an Ethernet component is accessing a secondary PIO. While the Ethernet component is accessing the secondary PIO, a processor can not access an SDRAM through a peripheral interface even if both the primary streaming output device and the peripheral interface are available.

According to various embodiments of the present invention, it is recognized that a bus is no longer required in certain applications such as system on a chip, system on a programmable chip, and other computer system implementations. A device such as a programmable logic device (PLD) or a field programmable gate array (FPGA) using a hardware descriptor language (HDL) is herein referred to as a programmable chip or a programmable device. Instead of implementing complicated bus sharing schemes using mechanisms such as splitting, the bus itself can be eliminated to improve system performance.

According to specific embodiments, it is recognized that primary components and secondary components need not be routed through a construct such as a bus. By not routing signals through a bus, a streaming output device can be implemented in a much more efficient manner. The conventional data and address lines that make up the bus are no longer the resource in contention. Instead, secondary components are resources in contention, as the number of physical lines connected to each secondary component in a system can be fixed. Furthermore, by not using a bus, interconnection flexibility is enhanced. For example, an Ethernet component can be allocated a variety of ports for directly accessing a memory and the only resource in contention would be the memory.

Consequently, a system bus arbitrator associated with all the secondary components in a computer system is no longer needed. Instead, secondary components themselves that may be accessed by more than one primary component are assigned individual secondary side arbitrators. An arbitrator that corresponds to a specific secondary component accessible by more than one primary component is referred to herein as a secondary side arbitrator. In one embodiment, there is a secondary side arbitrator for each secondary component in a computer system. In other embodiments, there is a secondary side arbitrator for selected secondary components in a system. The techniques of the present invention recognize that Ethernet component support for high-level language programs can be more efficiently and effectively provided in a system by using secondary side arbitration.

Figure 8 is a diagrammatic representation showing one example of a system using secondary side arbitration, sometimes referred to as slave side arbitration, simultaneous multiple primary components, or simultaneous multiple masters. A system using individual arbitrators that correspond to individual secondary components accessible by more than one primary component is referred to herein as a secondary side arbitration system. The secondary side arbitration system no longer requires a bus or a system bus arbitrator that prevents a second primary component from accessing a second secondary component when a first primary component is accessing a first secondary component. According to various embodiments, a secondary component such as peripheral interface 825 is associated with a secondary side arbitrator 851. However, secondary components UART 821 and PIO 823 are not associated with any arbitrator. In one example, secondary component UART 821 and secondary PIO 823 can only be accessed by primary CPU 813 and not by primary Ethernet device 815. A secondary memory component 825, however, can be accessed by both primary CPU 813 and primary Ethernet device 815.

According to various embodiments, a secondary side arbitrator 851 allows a first secondary component in a system to be accessed by a first primary component at the same time a second secondary component in the system is accessed by a second primary component. For example, peripheral interface 825 can be accessed by primary Ethernet 815 through secondary side arbitrator 851 at the same time, secondary UART 821 is accessed by primary CPU 813.

By allowing a CPU to access a secondary component at the same time another primary component such as a streaming output device or an Ethernet component is accessing memory, bus bottlenecks can be reduced. By using the simultaneous multiple primary component architecture, more direct connections between components can also be supported.

Figure 9 is a diagrammatic representation showing implementation of a system on a programmable chip. An input stage 901 receives selection information typically from a user for logic such as a processor core as well as other components such as a streaming output device to be implemented on an electronic device. In one

example, the input received is in the form of a high-level language program. A generator program 905 creates a logic description and provides the logic description along with other customized logic to any of a variety of synthesis tools, place and route programs, and logic configuration tools to allow a logic description to be implemented on an electronic device.

In one example, an input stage 901 often allows selection and parameterization of components to be used on an electronic device. The input stage 901 also allows configuration of variable or fixed latency support. In some examples, components provided to an input stage include intellectual property functions, megafunctions, and intellectual property cores. The input stage 901 may be a graphical user interface using wizards for allowing efficient or convenient entry of information. The input stage may also be a text interface or a program reading a data file such as a spreadsheet, database table, or schematic to acquire selection information. The input stage 901 produces an output containing information about the various modules selected.

In typical implementations, the generator program 905 can identify the selections and generate a logic description with information for implementing the various modules. The generator program 905 can be a Perl script creating HDL files such as Verilog, Abel, VHDL, and AHDL files from the module information entered by a user. In one example, the generator program identifies a portion of a high-level language program to accelerate. The other code is left for execution on a processor core. According to various embodiments, the generator program 905 identifies pointers and provides ports for each pointer. One tool with generator program capabilities is System on a Programmable Chip (SOPC) Builder available from Altera Corporation of San Jose, CA. The generator program 905 also provides information to a synthesis tool 907 to allow HDL files to be automatically synthesized. In some examples, a logic description is provided directly by a designer. Hookups between various components selected by a user are also interconnected by a generator program. Some of the available synthesis tools are Leonardo Spectrum, available from Mentor Graphics Corporation of Wilsonville, Oregon and Synplify available from Synplicity Corporation of Sunnyvale, California. The HDL files may contain

technology specific code readable only by a synthesis tool. The HDL files at this point may also be passed to a simulation tool 909.

As will be appreciated by one of skill in the art, the input stage 901, generator program 905, and synthesis tool 907 can be separate programs. The interface between the separate programs can be a database file, a log, or simply messages transmitted between the programs. For example, instead of writing a file to storage, the input stage 901 can send messages directly to the generator program 905 to allow the generator program to create a logic description. Similarly, the generator program can provide information directly to the synthesis tool instead of writing HDL files. Similarly, input stage 901, generator program 905, and synthesis tool 907 can be integrated into a single program.

A user may select various modules and an integrated program can then take the user selections and output a logic description in the form of a synthesized netlist without intermediate files. Any mechanism for depicting the logic to be implemented on an electronic device is referred to herein as a logic description. According to various embodiments, a logic description is an HDL file such as a VHDL, Abel, AHDL, or Verilog file. A logic description may be in various stages of processing between the user selection of components and parameters to the final configuration of the device. According to other embodiments, a logic description is a synthesized netlist such as an Electronic Design Interchange Format Input File (EDF file). An EDF file is one example of a synthesized netlist file that can be output by the synthesis tool 907.

A synthesis tool 907 can take HDL files and output EDF files. Tools for synthesis allow the implementation of the logic design on an electronic device. Some of the available synthesis tools are Leonardo Spectrum, available from Mentor Graphics Corporation of Wilsonville, Oregon and Synplify available from Synplicity Corporation of Sunnyvale, California. Various synthesized netlist formats will be appreciated by one of skill in the art.

A verification stage 913 typically follows the synthesis stage 907. The verification stage checks the accuracy of the design to ensure that an intermediate or final design realizes the expected requirements. A verification stage typically includes simulation tools and timing analysis tools. Tools for simulation allow the application of inputs and the observation of outputs without having to implement a physical device. Simulation tools provide designers with cost effective and efficient mechanisms for both functional and timing verification of a design. Functional verification involves the circuit's logical operation independent of timing considerations. Parameters such as gate delays are disregarded.

Timing verification involves the analysis of the design's operation with timing delays. Setup, hold, and other timing requirements for sequential devices such as flip-flops are confirmed. Some available simulation tools include Synopsys VCS, VSS, and Scirocco, available from Synopsys Corporation of Sunnyvale, California and Cadence NC-Verilog and NC-VHDL available from Cadence Design Systems of San Jose, California. After the verification stage 913, the synthesized netlist file can be provided to physical design tools 919 including place and route and configuration tools. A place and route tool typically locates logic cells on specific logic elements of a target hardware device and connects wires between the inputs and outputs of the various logic elements in accordance with logic required to implement an electronic design. The device can also be physically tested at 923.

For programmable logic devices, a programmable logic configuration stage can take the output of the place and route tool to program the logic device with the user selected and parameterized modules. According to various embodiments, the place and route tool and the logic configuration stage are provided in the Quartus Development Tool, available from Altera Corporation of San Jose, California. As will be appreciated by one of skill in the art, a variety of synthesis, place and route, and programmable logic configuration tools can be tested using various techniques of the present invention.

As noted above, different stages and programs can be integrated in a variety of manners. According to one embodiment, the input stage 901, the generator program

905, the synthesis tool 907, the verification tools 913, and physical design tools 919 are integrated into a single program. The various stages are automatically run and transparent to a user. The program can receive the user selected modules, generate a logic description depicting logic for implementing the various selected modules, and implement the electronic device. As will be appreciated by one of skill in the art, HDL files and EDF files are mere examples of a logic description. Other file formats as well as internal program representations are other examples of a logic description.

Figure 10 illustrates a typical computer system that can be used to implement a programmable chip having shared I/O lines. The computer system 1000 includes any number of processors 1002 (also referred to as central processing units, or CPUs) that are coupled to devices including memory 1006 (typically a random access memory, or "RAM"), memory 1004 (typically a read only memory, or "ROM"). The processors 1002 can be configured to generate a test sequences for any designated processor. As is well known in the art, memory 1004 acts to transfer data and instructions uni-directionally to the CPU and memory 1006 is used typically to transfer data and instructions in a bi-directional manner.

Both of these memory devices may include any suitable type of the computer-readable media described above. A mass storage device 1008 is also coupled bi-directionally to CPU 1002 and provides additional data storage capacity and may include any of the computer-readable media described above. The mass storage device 1008 may be used to store programs, data and the like and is typically a secondary storage medium such as a hard disk that is slower than memory. The mass storage device 1008 can be used to hold a library or database of prepackaged logic or intellectual property functions, as well as information on generating particular configurations. It will be appreciated that the information retained within the mass storage device 1008, may, in appropriate cases, be incorporated in standard fashion as part of memory 1006 as virtual memory. A specific mass storage device such as a CD-ROM 1014 may also pass data uni-directionally to the CPU.

CPU 1002 is also coupled to an interface 1010 that includes one or more input/output devices such as such as video monitors, track balls, mice, keyboards,

microphones, touch-sensitive displays, transducer card readers, magnetic or paper tape readers, tablets, styluses, voice or handwriting recognizers, or other well-known input devices such as, of course, other computers. Finally, CPU 1002 optionally may be coupled to a computer or telecommunications network using a network connection as shown generally at 1012. With such a network connection, it is contemplated that the CPU might receive information from the network, or might output information to the network in the course of performing the above-described method steps. It should be noted that the system 1000 may also be associated with devices for transferring completed designs onto a programmable chip. The above-described devices and materials will be familiar to those of skill in the computer hardware and software arts.

The hardware elements described above may be configured (usually temporarily) to act as multiple software modules for performing the operations of this invention. For example, instructions for running a generator program, input stage (e.g., a wizard), and/or compiler may be stored on mass storage device 1008 or 1014 and executed on CPU 1008 in conjunction with primary memory 1006.

Although many of the components and processes are described above in the singular for convenience, it will be appreciated by one of skill in the art that multiple components and repeated processes can also be used to practice the techniques of the present invention.

While the invention has been particularly shown and described with reference to specific embodiments thereof, it will be understood by those skilled in the art that changes in the form and details of the disclosed embodiments may be made without departing from the spirit or scope of the invention. For example, embodiments of the present invention may be employed with a variety of primary and secondary components and should not be restricted to the ones mentioned above. It is therefore intended that the invention be interpreted to include all variations and equivalents that fall within the true spirit and scope of the present invention.